COBB

Tips & techniques for MS-DOS and PC-DOS

November 1990
Vol. 1 No. 7

Inside DOS™

Redefining your function keys with ANSI.SYS

Do you know that DOS provides a device driver called ANSI.SYS that lets you completely take charge of your screen and keyboard? Thanks to ANSI.SYS, you can change the color of the display, change the position of the cursor on the screen, change the characters assigned to keys, and much more.

In this article, we'll show you how to use ANSI.SYS to assign custom commands to your function keys. As you'll see, the ability to assign a particular command or a series of commands to a function key can save you lots of time and keystrokes. In a future issue, we'll show you how to use ANSI.SYS to customize the colors on your screen.

Loading ANSI.SYS

Before you can redefine your function keys, you need to load ANSI.SYS. (By the way, ANSI is an acronym for American National Standards Institute—the organization that maintains standards for manipulating the keyboard and screen.) To load ANSI.SYS, simply place the line

```
device=c:\dos\ansi.sys
```

in your CONFIG.SYS file. (Of course, if you've installed your DOS files in a directory other than C:\DOS, you'll need to use that directory name in place of *c:\dos*.) Because DOS carries out the instructions in CONFIG.SYS only at startup, it won't actually install ANSI.SYS until you reboot your machine.

Once you've loaded ANSI.SYS, you can redefine a function key by entering the command

```
prompt $e[0;scancode;"F#";13p
```

where *scancode* is the scan code (59 through 68) of the function key you want to redefine and # is the number of that function key (1 through 10). Once you've issued a command that follows this form, pressing the [F#] key will tell DOS to run the batch file named F#.BAT. Table A lists the scan codes for the function keys [F1] through [F10].

You'll need to remember a couple of important points whenever you use the PROMPT command to redefine a function key. First, because the form of the PROMPT command we've shown you wipes out your system prompt, you'll probably want to issue an additional PROMPT command, like *prompt pg*, to restore your usual prompt. In addition, if you want to redefine more than one function key by placing several PROMPT commands in a batch file, make sure you precede those commands with the command

```
echo on
```

If you fail to turn on echo before DOS carries out your PROMPT commands, ANSI.SYS won't recognize those commands and will not redefine your keys.

Continued on page 12

Table A

Key	Scan code
[F1]	59
[F2]	60
[F3]	61
[F4]	62
[F5]	63
[F6]	64
[F7]	65
[F8]	66
[F9]	67
[F10]	68

Substitute the appropriate scan code for the scancode *argument in the command* prompt $e(0;*scancode*;"F#";13p.

IN THIS ISSUE

- Redefining your function keys with ANSI.SYS 1
- XCOPY's Verify option vs. the COMP command 2
- Using SUBST in AUTOEXEC.BAT 3
- Alternative ways to back up files 4
- Creating a special-purpose prompt 6
- Setting up a menu system with simple DOS commands 7
- Safeguarding AUTOEXEC.BAT from installation programs 9
- Quickly appending a directory to your path 10

COBB

Inside DOS™

Inside DOS (ISSN 1049-5320) is published monthly by The Cobb Group, Inc., 9420 Bunsen Parkway, Suite 300, Louisville, KY 40220. Domestic subscriptions: 12 issues, $39. Foreign: 12 issues, $59. Single issues: $4 domestic, $6.50 foreign. Send subscriptions, fulfillment questions, and requests for bulk orders to Customer Relations, 9420 Bunsen Parkway, Suite 300, Louisville, KY 40220, or call (800)223-8720 or (502)491-1900. Our FAX number is (502)491-4200. Address correspondence and special requests to The Editor, *Inside DOS*, at the address above.

POSTMASTER: Send address changes to The Cobb Group, Inc., P.O. Box 35160, Louisville, KY 40232. Second class postage is paid in Louisville, KY.

Editor-in-Chief:	Mark W. Crane
Technical Consultant:	Tim Landgrave
Contributing Editor:	Van Wolverton
Editing:	Jim Welp
	Toni Frank Bowers
Production:	Eric Paul
Design:	Karl Feige
Managing Editor:	Katherine Chesky
Publisher:	Douglas Cobb

Conventions

To avoid confusion, we would like to explain a few of the conventions used in *Inside DOS*.

As you probably know, Microsoft has released several versions of DOS. Although most of the articles in this journal apply to all versions of DOS since 2.00, we'll sometimes point out a particular command or technique that requires a particular version. For simplicity, we'll refer to all 4.xx versions as version 4.

Typically, we'll use DOS' default prompt, C>, to represent the DOS prompt in our examples. If you have defined a custom DOS prompt, just assume that the C> represents your prompt.

When we instruct you to type something, those characters usually appear on a separate line along with the DOS prompt. The characters you type will appear in red, while the characters DOS displays will appear in black.

Occasionally, we won't display the command you type on a separate line. In these cases, we'll display the characters you type in italics. For example, we might say, "issue the command *dir *.txt* at the DOS prompt." Although DOS is not sensitive to capitalization, we'll always display the characters you type in lowercase.

When we refer to a general DOS command (not the command you actually type at the DOS prompt), we'll display that command name in all caps. For example, we might say, "you can use either the COPY or XCOPY command to transfer files from one disk to another."

Many commands accept parameters that specify a particular file, disk drive, or other option. When we show you the form of such a command, its parameters will appear in italics. For example, the form of the COPY command is:

```
copy file1 file2
```

where *file1* and *file2* represent the names of the source file and the target file, respectively.

The names of keys, such as [Shift], [Ctrl], and [F1], appear in brackets. The [Enter] key is represented by the symbol ↵. When two keys must be pressed simultaneously, those key names appear side by side, as in [Ctrl][Break] or [Ctrl]z.

LETTERS

XCOPY's Verify option vs. the COMP command

Q When I back up a file, I want to have absolute assurance that the backup is an exact duplicate of the original. So far, I've been using the COMP command to confirm the accuracy of the backup. First, I use XCOPY to place all the files in a directory, then I use COMP to compare all the copied files with the originals. Would I be better off using XCOPY's Verify option (/v) instead?

Harley E. Beard
Everett, Washington

A Mr. Beard is indeed better off using COMP instead of XCOPY's Verify option to ensure the integrity of his backup files. To understand why, you need to understand exactly how both the COMP command and the Verify option work.

How COMP works

The COMP (or Compare) command tells DOS to perform a byte-by-byte comparison of two files, and to let you know if those files are identical. If DOS finds a mismatch, it displays a message indicating the location of the mismatch and the hexadecimal value of the differing bytes in each file. Although you'll typically have no use for these hexadecimal values, the COMP command certainly allows you to determine whether two files are duplicates.

If the files are identical, DOS will display the message

```
Files compare OK
```

and will ask if you want to compare any more files. If the files are not the same length, DOS will display the message

```
Files are different sizes
```

Finally, if the files are the same length but do not have the same content, DOS will display the bytes that differ. If more than 10 bytes differ, DOS will display the message

```
10 Mismatches - ending COMP
```

and will end the COMP procedure.

The form of the COMP command is

```
comp file1 file2
```

where *file1* and *file2* are the names of the two files you want to compare. You can include a drive and a path in *file1* and *file2*, and you can use wildcard characters to specify groups of files with similar names.

If you want, you can execute the COMP command by simply entering

```
C>comp
```

and allowing DOS to prompt you for the necessary information.

By the way, COMP is an external DOS command—DOS cannot execute it unless it can find the file COMP.COM. Therefore, make sure your path includes the directory that contains your DOS files, or you'll probably see the message *Bad command or file name* when you issue the COMP command.

How the Verify (/v) option works

Unlike the COMP command, XCOPY's Verify option (/v) doesn't perform a byte-by-byte comparison of two files. Instead, it appends an error-detection number called a Cyclical Redundancy Check (CRC) to each file it copies. The CRC is a type of "check sum" that is based on all of the bytes stored in the file. If two files are identical, they will produce the same CRC value. Therefore, if XCOPY copies a file, then finds that the CRC values are the same for both the old and new versions of that file, it will assume that the copied file is error-free. If the CRC values of the two files don't match, however, XCOPY will recopy the file and recheck its CRC value, then, if necessary, produce an error message.

For example, if you enter

```
C>xcopy a:memo.txt b: /v
```

DOS copies the file MEMO.TXT from drive A to drive B, then verifies that both versions of this file have the same CRC value.

By the way, DOS' VERIFY command allows you to turn on the verify feature, which performs CRC error-checking each time you save a file to disk. To turn the verify feature on, simply type

```
C>verify on
```

To turn the verify feature off (the default setting), just type

```
C>verify off
```

Finally, if you type the VERIFY command with no parameters (just enter *verify*), DOS will display the current status of the verify feature (*VERIFY is on* or *VERIFY is off*).

Using SUBST in AUTOEXEC.BAT

Q I found the article "Shortening Directory Paths Using the SUBST Command" in your September issue very helpful. However, you failed to point out one important matter: If you precede the PATH command in your AUTOEXEC.BAT with a SUBST command, you need to type the full path name of the file SUBST.COM so that DOS will know where that file is stored. For instance, in your article, you suggested placing the following commands in AUTOEXEC.BAT

```
subst k: c:\apps
path c:\dos;k:\123;k:\wp;k:\hg
```

Unless you've copied the file SUBST.COM into the root directory, however, DOS won't be able to carry out the SUBST command, and will generate the error message *Bad command or file name*. Additionally, when DOS attempts to carry out the PATH command, it will generate the message *Invalid command in path*.

To correct the problem, all you have to do is precede the SUBST command with the directory where the file SUBST.COM is stored, like this:

```
c:\dos\subst k: c:\apps
path c:\dos;k:\123;k:\wp;k:\hg
```

This allows DOS to successfully make the correct drive and directory substitution, and therefore to set up the desired path.

Bud Case
Rochester, New Hampshire

A We thank Mr. Case for identifying this important oversight, and for taking time to notify us of this problem. For his efforts, he'll receive $25.

VAN WOLVERTON

This article was written by Contributing Editor Van Wolverton. Van is the author of the bestselling books Running MS-DOS *and* Supercharging MS-DOS.

Alternative ways to back up files

Just as you know you should eat sensibly, floss, and exercise regularly, you know you should back up the files on your fixed disk. All the guilt-inducing warnings about backing up files can be summarized by paraphrasing one dentist's comment on flossing: You don't have to back up *all* your files, just the ones you'd like to keep.

Two familiar road blocks stand in the way of regular file backup: time and money. No matter how you back up your files, it takes time to set up a backup procedure, and more time whenever you back up. Depending on the hardware and software you choose, you can spend several hundred to several thousand dollars trying to reduce the time required, but it isn't necessary to spend anything. We'll look at some of the hardware and software alternatives, then see how to use DOS itself to set up a serviceable, reasonably quick backup procedure.

Hardware alternatives

One way to describe the capacity of a fixed disk is to recite how many diskettes—preferably 360K diskettes—it would take to hold the same amount of data (as in "You'd need almost 57 diskettes to store the files on a 20-megabyte fixed disk"). How often would you back up your fixed disk if you had to swap 57 diskettes every time? The picture isn't really that bleak, as you'll see in a moment, but there's no escaping the fact that using some other storage device is more convenient than swapping diskettes.

You've got several hardware choices for alternative backup devices: a cartridge tape drive, an optical disk drive, or even another fixed disk.

Cartridge tape drives typically store 40 to 120 megabytes and cost $200-$600; most of them include software that lets you back up selected files or simply copy the entire fixed disk to tape (which obviously takes longer). Tape drives based on digital audio tape (DAT) technology are just becoming available. They boast very high capacity (hundreds of megabytes) and corresponding prices (thousands of dollars).

Optical disk drives use the same basic recording technology as audio compact discs (CDs). Like DAT tapes, they have very high capacity (500+ megabytes) and are very pricey ($3000 to $6000 or more, depending on whether you can erase data once you have written it).

The simplest hardware solution is to install two fixed disks and use one to back up the other. Depending on the size of your second fixed disk, this can cost only a few hundred dollars.

Software alternatives

If you don't want to buy additional hardware, you'll have to back up to diskettes. To reduce both the time and number of disk swaps, you can buy a backup program that compresses the backup files (so that more fit on each diskette) and uses special techniques to write the data more quickly on the diskette. (Some of the programs write so quickly that they fill the diskettes almost as fast as you can put them in the drive.) The more popular backup programs include Fastback Plus, PC Fullback, Norton Backup, and PC Tools Deluxe; they cost from $75 to $150.

All these hardware and software alternatives require time to set up your backup procedure. In addition to their cost, they all (except the additional fixed disk) share another disadvantage: Because of the way they store data on the tape or optical disk, it's difficult or impossible to work with the backed up files using standard DOS commands; you've got to use the backup program or the software that came with the hardware to access the backed-up files.

DOS can do the job

The major reason for these backup alternatives is the inconvenience of—and potential for error in—dealing with dozens or hundreds of backup diskettes. But unless you routinely work with many large files (half a megabyte or more) or you're the manager of a network, it really isn't necessary to buy additional hardware or software for backing up your files. DOS includes a BACKUP command to be used expressly for backing up files and, as a couple of recent letters to the editor pointed out, the XCOPY command can be an even more useful alternative. But what about all those backup diskettes?

You don't have to deal with a drawer full of backup diskettes because, as you'll remember, you don't have to back up *all* the files on your fixed disk, just the ones you'd like to keep (or, more accurately, just the ones you can't afford to lose). You've got the originals of all your program files, so you don't have to back them up; the same is true for data files you buy, such as clip art for desktop publishing. The only files you have to back up are the data files you create, such as word processing documents, spreadsheets, and database files. And you don't have to back up each of these files every day, just those you have changed.

It shouldn't take more than an hour or two to set up your backup procedure, and just a few minutes each day

to back up the files. But first, you have to decide whether to use the BACKUP or XCOPY command.

The BACKUP command

The BACKUP command is intended specifically for backing up files from a fixed disk to diskettes. Early versions weren't especially useful but, like DOS, BACKUP has improved with age. Its advantages are that it formats diskettes, compresses the backup files, and can handle a file larger than the backup diskette. The last advantage is the most important: If you have to back up files larger than your backup diskettes, you must use BACKUP.

However, BACKUP has several disadvantages:

- It can't replace a single backup file on a backup diskette—it either erases all files on the backup diskette or adds files to those already there.
- You can't work with the backed-up files using regular DOS file commands, you can only copy them back to the fixed disk with the RESTORE command.
- In most cases, you can't restore files that were backed up with a different version of DOS.

The XCOPY command

The XCOPY command is a modern alternative to the COPY command introduced with version 3.2 of DOS. Its advantages and disadvantages are mirror images of BACKUP's qualities: Files copied with XCOPY are normal DOS files, XCOPY replaces individual files if they already exist on the backup diskette, and XCOPY doesn't care about DOS versions. On the down side, XCOPY requires formatted diskettes, doesn't compress files, and can't copy a file larger than the backup diskette.

If you back up more than 50 files a day, you should probably use BACKUP to take advantage of its compression and somewhat faster operation. If any of the files you back up are larger than the capacity of your backup diskettes, of course, you must use BACKUP. But if you back up 50 or fewer files each day—and you should back up every day—XCOPY is probably the better choice; it replaces existing files on the backup diskette, reducing the proliferation of backup diskettes, and it's easier to manage your backup files if they're ordinary DOS files.

The BACKUP batch file

To make your daily backup procedure as painless as possible, put the XCOPY or BACKUP command in a batch file. When you're ready to shut down for the day, just make sure the backup diskettes are handy and type *bakup* (or whatever you call your batch file). It shouldn't take more than two or three minutes—probably the cheapest insurance you can buy.

The easiest backup procedure to manage involves first identifying the directories that contain data files you'll need to back up. Then creating a batch file that includes the XCOPY command's */m* parameter for each directory to back up the files that have changed since they were last backed up. You'll need time to put the proper backup diskette in the drive, so place a PAUSE command between each XCOPY command.

For example, suppose that you use Microsoft Word and Microsoft Excel; your Word documents are stored in several subdirectories of \WORD; and all the spreadsheet and macro files are stored in \EXCEL. You want to back up all files in the \WORD directory and its subdirectories whose extension is DOC, and all files in the \EXCEL directory whose extension is XLS or XLM. Figure A shows a sample batch file that will perform this operation for you.

Figure A

```
@echo off
cls
echo Put WORD backup disk in drive A
pause
xcopy c:\word\*.doc /s /m /v
echo Put EXCEL backup disk in drive A
pause
xcopy c:\excel\*.xl? /m /v
```

This batch file tells DOS to back up the files in the \WORD and \EXCEL directories.

The XCOPY command for \WORD includes the */s* parameter because the \WORD directory contains subdirectories with files to be backed up, and both XCOPY commands include the */v* (for *verify*) parameter to ensure that the backup copy is accurate. Add ECHO, PAUSE, and XCOPY commands for each directory that you want to back up. (If you're going to back up a directory to the backup diskette already in drive A, you can omit the ECHO and PAUSE commands.)

As the letter in last month's issue pointed out, the */m* parameter tells XCOPY (or BACKUP) to copy all files whose archive attribute is on, then turns off the archive attribute. It probably isn't necessary to use the ATTRIB command to turn on the archive attribute of all your files since DOS creates all files with the archive attribute on, and only BACKUP, XCOPY, and the commercial backup programs turn it off. It doesn't hurt, though.

If XCOPY tells you there is insufficient disk space, put a formatted diskette in drive A and run the batch file again. Label the diskette as the second (or third or whatever) backup diskette for \WORD (or whichever directory was being backed up).

Creating a special-purpose prompt

Many application programs like Lotus 1-2-3 and WordPerfect let you go to the DOS prompt, execute DOS commands, and return to the application at the same point you left it. Although this is a convenient feature, it does present the following problem: When you temporarily go to DOS, it's easy to forget that your application is still active. If you do forget, and restart the same application or turn off your computer, you lose the chance to save the file that was active when you went to DOS and therefore you lose all the work you've done since you last saved that file.

Fortunately, you can write a simple batch file that creates a custom prompt reminding you to go back to your application, then restores the prompt to normal when you exit the application. In this article, we'll show you how to write and use that batch file.

How the "go-to-DOS" feature works

When you select the "go-to-DOS" feature from within an application, you never actually exit from the application (although it looks like you do). Instead the application executes the file COMMAND.COM, the DOS command processor. At this point, the DOS command processor resides within the application. Using this copy of COMMAND.COM, you can execute DOS commands while your application remains active. When you're finished using DOS, you usually type *exit* and press ↵ to return to the application.

The special-purpose prompt

A batch file like the one shown in Figure A creates a special reminder prompt and then loads your application. You'll see the reminder prompt only when you go to DOS using the application's go-to-DOS feature. When you quit from the application, you'll see your normal DOS prompt once again. The last line of the batch file, which restores the normal prompt, is not executed until you exit completely from the application.

Figure A

```
@echo off
prompt Enter EXIT to return to 123$_$p$g
cd \123
123
prompt $p$g
```

This batch file changes the DOS prompt to remind you to go back to your application.

An example

The batch file in Figure A is named LOTUS22.BAT and is designed to work with Lotus 1-2-3 Release 2.2. Let's look at the file line by line to see how it works.

The first line of LOTUS22.BAT issues the command

```
@echo off
```

which tells DOS to suppress the display of the batch file commands as it executes them. The second line

```
prompt Enter EXIT to return to 123$_$p$g
```

creates a two-line DOS prompt. The special code $_ (underscore), which follows the text *Enter EXIT to return to 123*, tells DOS to start a new line. The special codes *$p* and *$g* tell DOS to display the current drive and directory followed by a greater than sign. The resulting prompt looks like this:

```
Enter EXIT to return to 123
C:\123>
```

The third and fourth lines of LOTUS22.BAT,

```
cd \123
123
```

tell DOS to change to the \123 directory and load 1-2-3. Of course, we could omit the third line if the \123 directory were included in our path.

The last line

```
prompt $p$g
```

restores the normal prompt. Although you can define any prompt you want, ours includes the current drive and directory followed by the greater than sign (>).

Modifying the batch file for other applications

You'll need to write a separate batch file for each application that has a go-to-DOS feature. Fortunately, it's easy to modify our simple batch file to work with other applications.

Of course, regardless of the application, the first line of each batch file stays the same. In the second line, you'll want to reword the text in the custom prompt to whatever is appropriate for your application. Beginning with the third line, you should supply the commands needed to load your application. Of course, you'll need to use one

command per line. Finally, you'll need to modify the last line of your batch file if you want to restore a custom DOS prompt that's different from the one we've chosen.

You'll want to give your custom batch file a root name that describes the application it launches. For instance, if you're creating a batch file for WordPerfect 5.1, you might want to name the file WP51.BAT.

Using the batch files

Don't forget to save your new batch files in a directory that's included in your path, so DOS can find them when you're ready to execute them. That way, whenever you want to load an application, just enter the name of the associated batch file at the DOS prompt.

If you normally use a menu program to select and load your applications, the menu will probably work just fine with your batch files. Instead of specifying the usual application startup commands in the menu program, specify the batch file name.

By the way, you'll want to avoid performing a couple of potentially harmful tasks while working in an application's go-to-DOS mode. First, and most importantly, don't run any programs that terminate and stay resident (TSRs). These programs "glue" themselves into a portion of memory and will not allow you to return to your application. In addition, be careful not to delete any of the application's temporary files while working in go-to-DOS mode. Otherwise, the application may not run properly (or at all) when you try to return.

BATCH FILE TECHNIQUE

Setting up a menu system with simple DOS commands

When you were in school, did you prefer exams that asked "fill-in-the-blank" questions, or those that had multiple-choice questions? Unless you were the type of person who eagerly looked forward to extra credit projects, you probably preferred the simplicity of multiple-choice exams. Rather than having to pencil in answers off the top of your head for fill-in-the-blank exams, all you needed to do on a multiple-choice test was choose an answer from a short list.

Just as it is easier to answer a question by choosing from a list, it's easier to launch an application by choosing an option from a menu. For this reason, many people set up menu systems on their computers that let them view menus of available applications, and then launch one by pressing a single key.

As you may know, there are several ways to set up a menu on your system. First, you can buy an inexpensive off-the-shelf menu program. Alternatively, you can set up a custom menu system using a collection of relatively simple batch files. Perhaps the best solution, though, is to set up a menu system using the techniques explained in the article on page 1. In this article, we'll show you how to apply those techniques to set up a simple, but effective menu system.

Creating the menu

Let's suppose you've installed three applications on your computer—WordPerfect, Lotus 1-2-3, and Harvard Graphics—and that you want to create a menu for launching these applications. The first step is to create a text file named MENU.TXT that displays the menu of available applications along with the function key you'll use to launch each one. After you've created MENU.TXT, store it in a directory named C:\MENU.

For example, Figure A shows a simple menu that will get the job done for WordPerfect, 1-2-3, and Harvard Graphics. If you want, of course, you can jazz up your menu with fancy characters and borders to give the menu a real "professional" look. (We'll discuss entering special characters in detail in a future issue.)

Figure A

```
Press... to launch...

F8        WordPerfect

F9        Lotus 1-2-3

F10       Harvard Graphics

(To refresh the menu, press F1)
```

We've placed the text we want to appear as the menu in a file named MENU.TXT.

Redefining the function keys

Once you've created the menu, you need to use the technique we explain in the lead article to redefine the function keys listed on the menu. In our case, then,

we need to redefine the function keys F1, F8, F9, and F10. To do this, enter into the batch file REDEFKEY.BAT the commands

```
echo on
prompt $e[0;59;"F1";13p
prompt $e[0;66;"F8";13p
prompt $e[0;67;"F9";13p
prompt $e[0;68;"F10";13p
echo off
```

The first command, *echo on*, allows DOS to send the commands in the batch file to ANSI.SYS. The second command tells DOS to run the batch file F1.BAT when you press the F1 key; the third command tells DOS to run the batch file F8.BAT when you press the F8 key; the fourth command tells DOS to run the batch file F9.BAT when you press the F9 key; and the fifth command tells DOS to run the batch file F10.BAT when you press the F10 key. (By the way, these PROMPT commands will work successfully only if you've loaded the device driver ANSI.SYS, as explained in the lead article.) Finally, the command *echo off* tells DOS to suppress the display of any additional batch file commands.

In addition to entering PROMPT commands into REDEFKEY.BAT, you need to write the batch files that are associated with each function key: F1.BAT, F8.BAT, F9.BAT, and F10.BAT. Because you want the F1 key to display the menu on the screen, you'll want to enter into F1.BAT the commands

```
@echo off
cls
type c:\menu\menu.txt
```

(If you're using DOS version 3.2 or earlier, omit the @ symbol in the *@echo off* command.) Similarly, because you want the F8 key to launch WordPerfect, you'll want to enter the following commands into F8.BAT, which launch WordPerfect and then redisplay the menu:

```
@echo off
cls
cd \wp
wp
cd \
type c:\menu\menu.txt
```

Follow this same formula for creating both the F9.BAT and F10.BAT files, which launch 1-2-3 and Harvard Graphics, respectively. Once you've created all of your F?.BAT files, it's essential for you to store all of those files along with MENU.TXT and REDEFKEY.BAT in the directory C:\MENU, and to add that directory's name to your PATH statement. Otherwise, DOS won't be able to find the commands associated with the function keys when you press those keys.

Using the menu system

Now that you've created your menu and redefined your function keys, you're ready to try out your menu system. To redefine your function keys and bring up the menu when your computer restarts, you'll need to append the following three lines to the very bottom of your AUTOEXEC.BAT file:

```
call c:\menu\redefkey.bat
cls
type c:\menu\menu.txt
```

If you're using DOS 3.2 or earlier, you'll need to use the command

```
command \c c:\menu\redefkey.bat
```

instead of

```
call c:\menu\redefkey.bat
```

Now, when you reboot your computer, your custom menu will appear on the screen, like this:

```
Press... to launch...

F8        WordPerfect

F9        Lotus 1-2-3

F10       Harvard Graphics

(To refresh the menu, press F1)

C>_
```

You can now simply look at the menu and press the function key that launches the desired application. For instance, to launch WordPerfect, just press F8. To launch 1-2-3, press F9. It's that simple! When you exit from any of the selected applications, DOS will refresh the menu. Of course, you can refresh the menu at any time while you're working at the DOS prompt simply by pressing the F1 key.

Conclusion

Because it is easier to launch applications from a menu, you might want to set up a menu system on your computer that allows you to launch applications with a single function key. In this article, we've applied the concepts presented in the lead article and outlined the process for setting up a menu system using simple DOS tools.

Safeguarding AUTOEXEC.BAT from installation programs

The idea for this article was submitted by Brian Livingston of New York, New York.

If you've recently installed any new software packages, you probably know that most packages attempt to make the installation process easy by providing an installation program (usually called *Install* or *Setup*). Many installation programs will, with or without your permission, make significant revisions to your AUTOEXEC.BAT file. While these changes typically optimize the performance of the newly installed application, they may sometimes play havoc elsewhere in your system. In this article, we'll show you a simple technique that safeguards you from an installation program's unwanted changes.

What do installation programs do?

Installations programs generally make two types of changes to AUTOEXEC.BAT:

- inserting a directory into the PATH statement
- appending new commands to the end of the file that load memory-resident programs (that is, programs that load and stay in memory until you reboot)

Let's consider the impact of each of these changes.

In order to maximize your system's overall performance, your PATH statement should list the directories most frequently accessed first, and list those least frequently accessed last. Therefore, if an installation program inserts a new, infrequently accessed directory name at the very beginning of your path, you'll want to move that directory name to the end of the list.

As you may have discovered, memory-resident programs can be quite picky about the order in which DOS loads them into memory. Some memory-resident programs insist on being loaded first, or last, or before or after a particular application. Therefore, since many installation programs append lines to AUTOEXEC.BAT that load memory-resident programs, you may need to adjust the order in which these lines appear, or remove them altogether.

The safeguard technique

Fortunately, there's a simple way to keep installation programs from tampering with your AUTOEXEC.BAT. First, place the startup commands you'd normally place in AUTOEXEC.BAT in another batch file named START.BAT. Then, create an AUTOEXEC.BAT file that contains only the following two commands:

```
@echo off
C:\start.bat
```

Once you've created START.BAT and this "lean" AUTOEXEC.BAT, you'll never fear an installation program again. When an installation program asks you if it may modify AUTOEXEC.BAT, you can confidently answer "yes." When the installation program is finished, you can go back and examine the changes that were made, and either incorporate those changes into your START.BAT file, or delete them altogether. Finally, once you've correctly modified START.BAT, you can trim AUTOEXEC.BAT back down to its original two commands.

By the way, using this technique offers one additional benefit: If you forget (or don't have time) to examine the changes made to AUTOEXEC.BAT immediately after running the installation program, DOS still won't recognize those changes when you reboot. Here's why: Installation programs always add commands they deem necessary to the very *end* of AUTOEXEC.BAT. Because the second line of our AUTOEXEC.BAT invokes START.BAT without a CALL or COMMAND /C prefix, DOS won't return to AUTOEXEC.BAT after it runs START.BAT, and therefore won't execute the new commands.

Conclusion

In this article, we've shown you a technique that makes it easy to manage the changes that installation programs propose for your AUTOEXEC.BAT file. Essentially, all you have to do is keep your startup commands in a separate batch file named START.BAT, then invoke that batch file with a line in AUTOEXEC.BAT.

Quickly appending a directory to your path

The following article was submitted by Katherine Chesky. Katherine is the Managing Editor for The Cobb Group's Applications Group.

In a previous issue of *Inside DOS*, we showed you how to include a PATH statement in your AUTOEXEC.BAT file ("Getting to Know a Very Important File: AUTOEXEC.BAT," July 1990). As you probably know, the PATH command helps DOS find the directories containing DOS commands and application programs. Typically, when you install a new application program, you'll need to revise the PATH statement in your AUTOEXEC.BAT file to include the new directory.

Sometimes, though, you may not want to modify your PATH statement. Perhaps there's a program you seldom use, and you don't want to slow down your computer system by keeping the program's directory name permanently in your path. Or perhaps you're trying out a new program that you may not keep on your machine.

In this article, we'll build a batch file that allows you to easily append a directory to your existing path. We'll also build a batch file that makes it just as easy to restore your original path.

A few basics

Before we show you our path-modifying batch files, we need to make sure you're aware of a few essential concepts. First, one of our batch files will include the replaceable parameter *%1*. As you may recall from earlier issues of *Inside DOS*, DOS will replace the symbol *%1* with the parameter you supply at the DOS prompt along with the batch command name. For instance, if you have a batch file named GO.BAT, and you type the command

```
C>go today
```

DOS will substitute the word *today* for every occurrence of the %1 symbol that appears in GO.BAT.

In addition to a replaceable parameter, our batch files will use special variables called *environment variables*. To define an environment variable, you must use the following form of the SET command

```
set string1=string2
```

where *string2* is the string (or value) you want to assign to *string1*. For instance, if you want to assign the filename BUDGET.WK1 to the string *file*, you would use the command

```
set file=budget.wk1
```

Once you've defined an environment variable, you can access that variable in a batch file by enclosing it within percent signs (%). For instance, if you've issued the command *set file=budget.wk1*, and you want to copy BUDGET.WK1 to drive A, you can use the command

```
copy %file% a:
```

Finally, when you define a path using the PATH command, DOS stores that path as an environment variable named—amazingly enough—*path*. Therefore, you can always retrieve the current path from the environment using the string *%path%*.

Appending a directory to your path

Our ADDPATH.BAT file, which we show in Figure A, lets you append a directory to your existing path. To run this batch file, you'll type the command

```
addpath directory
```

where *directory* is the name of the directory you want to add to the path. Let's examine ADDPATH.BAT line by line.

Figure A

```
@echo off
if %1a==a goto syntax
set oldpath=%path%
path=%path%;%1
goto end
:SYNTAX
echo The form of this command is
echo ADDPATH <directory>
:END
```

The ADDPATH.BAT file lets you append a directory to your existing path.

As usual, the first line issues the command

```
@echo off
```

which tells DOS to suppress the display of the batch file commands as it executes them. The next line

```
if %1a==a goto syntax
```

checks to see if you typed a parameter when you executed the batch file. If you did (and you're supposed to), DOS continues to the third command in the batch file. However, if you didn't supply a parameter, then DOS carries out the command that immediately follows the line containing the label *:SYNTAX*.

As we just mentioned, if you supply a parameter to ADDPATH.BAT, DOS will carry out the third command in the batch file:

```
set oldpath=%path%
```

This command tells DOS to store the current path in an environment variable named *oldpath*. As you'll see when we discuss our second batch file, OLDPATH.BAT, we'll use the variable *oldpath* to restore our original path.

After storing the current path in the variable *oldpath*, DOS will execute the fourth command in the batch file:

```
path=%path%;%1
```

This command defines our new path. As we mentioned earlier, DOS substitutes the current path for the string *%path%*, and substitutes the parameter you supplied on the command line for the symbol *%1*. The new path, then, will be exactly what we want—the old path *plus* the directory you supplied as a parameter to ADDPATH.BAT.

The next command in the batch file

```
goto end
```

tells DOS to skip over the :SYNTAX section and to go to the label *:END* at the very end of the batch file. When DOS reaches the label *:END*, it will cease execution of ADDPATH.BAT, and will return you to the DOS prompt.

As we mentioned, when you execute the batch file ADDPATH.BAT without supplying a directory name, the IF statement in line 2 of the batch file will tell DOS to carry out the commands following the label *:SYNTAX*:

```
echo The form of this command is
echo ADDPATH <directory>
```

These commands display on the screen the message

```
The form of this command is
ADDPATH <directory>
```

which reminds you to supply a directory name on the command line. As we've mentioned, the label *:END*, which appears in the last line in the batch file, simply marks the end of the batch file, and is required because of the *goto end* statement in the file's fifth line.

An example

To illustrate how ADDPATH.BAT works, suppose you want to add a directory named *\wizbang* to your path so you can try out a new program. To do this, enter the command

```
C>addpath \wizbang
```

As soon as you press ↵, DOS will check the environment and retrieve the current path setting. Then it will append the directory *\wizbang* to the end of the path.

If you want, you can use the PATH command to make sure DOS added the additional directory. Just type *path* at the DOS prompt and DOS will display the current path. For example, if your original path was

```
\dos;\wp;\123
```

and you used ADDPATH.BAT to add *\wizbang*, then the new path will be

```
\dos;\wp;\123;\wizbang
```

Restoring your original path

Because ADDPATH.BAT doesn't modify the PATH statement in your AUTOEXEC.BAT file, you can restore your original path simply by rebooting your machine. However, since you'll probably want to restore your original path without rebooting, we've created a simple batch file named OLDPATH.BAT that restores the previous path. Figure B shows this batch file.

Figure B

```
@echo off
path=%oldpath%
```

The OLDPATH.BAT file restores your path to its original setting.

The first line of OLDPATH.BAT

```
@echo off
```

suppresses the display of the file's commands. The second line

```
path=%oldpath%
```

sets the path equal to the string stored in the variable *%oldpath%*. As you'll recall, our ADDPATH.BAT file stored the original path in the variable *oldpath*. Therefore, this line retrieves the original path and makes it the current path.

To execute the OLDPATH.BAT file, simply enter the command

```
C>oldpath
```

When you issue this command, all you'll see is the DOS prompt. If you want, though, you can use the PATH command to make sure DOS restored the original path.

A couple of drawbacks

Before you try our ADDPATH.BAT and OLDPATH.BAT files, keep in mind these two potential show stoppers: First, *%path%* only works in DOS version 3.2 or higher. In addition, you can't append a directory to the path if there's not enough room in your environment. If you run ADDPATH.BAT and see the message

```
Out of environment space
```

you'll need to increase the computer's environment space. We'll show you how to increase the computer's environment space in a future issue of *Inside DOS*.

Conclusion

In this article, we presented some batch files that make it easier to modify your path. The ADDPATH.BAT file provides an easy way to append a directory to your existing path without revising your AUTOEXEC.BAT file. On the other hand, the OLDPATH.BAT file lets you restore your previous path.

THE COBB GROUP
Postmaster:
Monthly Technical Journal
Please Rush!
Second Class Postage

Please include account number from label with any correspondence.

Redefining your functions keys
Continued form page 1

An example

For example, suppose you want to assign the command

```
dir /p
```

to the [F9] key. To do this, first determine the scan code for the [F9] key by looking in Table A—67. Next, issue the PROMPT command we showed you earlier, substituting *67* in place of *scancode*, and *9* in place of #, like this:

```
C>prompt $e[0;67;"F9";13p
```

This command tells DOS that whenever you press the [F9] key, it should run the file F9.BAT. At this point, issue the command

```
C>prompt
```

to restore the default system prompt: C>. Finally, create a file named F9.BAT that contains only the command

```
dir /p
```

followed by a return. Make sure you store F9.BAT in a directory whose name is on the path.

Once you've issued the PROMPT command and created F9.BAT, you're ready to try out your new function key by pressing [F9]. Immediately, DOS will display and carry out the command *dir /p*, just as if you'd typed the command manually and pressed ↵.

Another example

You'll probably find that redefined function keys are especially useful for issuing rather long or complicated commands. For instance, if you often need to format 360 Kb diskettes in a 1.2 Mb floppy drive, you probably use the command

```
C>format a: /n:9 /t:40
```

To avoid typing the entire command each time you format a diskette, you can assign it to one of your function keys. To assign this command to the function key [F10], for instance, first determine the scan code for the [F10] key (68) by looking in Table A. Now, issue the PROMPT command we presented earlier, substituting 68 for *scancode* and 10 for #, like this:

```
C>prompt $e[0;68;"F10";13p
```

At this point, issue the command

```
C>prompt
```

to restore the default system prompt: C>. Finally, move into one of the directories on your path, and create a file named F10.BAT that contains only the command

```
format a: /n:9 /t:40
```

You can now use the [F10] key to format 360 Kb diskettes whenever you want. As soon as you press [F10], DOS will carry out the command *format a: /n:9 /t:40*, just as if you'd typed the command and pressed ↵.

Notes

As we mentioned earlier, if you want to redefine several function keys at once by entering multiple PROMPT commands in a batch file, make sure you precede those commands with the command *echo on*. Otherwise, ANSI.SYS won't recognize those commands and will not redefine your keys. Of course, after you've issued the PROMPT commands that redefine your keys, you'll probably want to use the command *echo off* to turn echo back off, and then use an additional PROMPT command to restore your system prompt.

How does this affect applications?

Fortunately, you shouldn't have to worry about redefined function keys fouling up your applications. Nearly all major applications ignore the key definitions you've made while you're working within that application. As soon as you exit the application, however, DOS will once again recognize the custom function keys you've defined.

Conclusion

In this article, we've explained how to redefine your function keys, and we've shown you a couple of examples that illustrate how useful redefined function keys can be. If you want to go a step further with redefined function keys, read the article that begins on page 7 entitled "Setting up a Menu System with Simple DOS Commands."